PAT-NO: JP410040087A

DOCUMENT-IDENTIFIER: JP 10040087 A

TITLE: METHOD FOR HANDLING DATA MODEL TO BE USED FOR SOFTWARE ENGINEERING

PUBN-DATE: February 13, 1998

INVENTOR-INFORMATION:
NAME
ELISABETH, LEPRINCE
ROBERT, CAROSSO
KISO, JAMES
EDWARD, STRASSBERGER

INT-CL (IPC): G06F009/06

ABSTRACT:

PROBLEM TO BE SOLVED: To provide a data model handling method excellent in transplantation between tools.

SOLUTION: The method executes data model handling for applying a data model to a conversion rule mode by using the transfer of a model among a plurality of tools Ta to Td and bridges Ba to Bd capable of converting the format of a certain tool into another format through an **intermediate format** NIM. At first, a **bridge** is prepared by approach called as object orientation based on execution language. The **bridge** is an executable instruction capable of uniformly or conditionally converting the data model in a plurality of steps to be continuously started and each step is an executable instruction of which code is developed, based on a reusable class assembly. A procedure is simply started only by providing the name and parameter of the procedure.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平10－40087

(43)公開日　平成10年(1998)２月13日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/06	５３０		Ｇ０６Ｆ　9/06	５３０Ｕ	

審査請求　有　　請求項の数４　ＯＬ　（全 32 頁）

(21)出願番号　特願平8－327058

(22)出願日　平成８年(1996)12月６日

(31)優先権主張番号　９５　１４５７６

(32)優先日　1995年12月８日

(33)優先権主張国　フランス（ＦＲ）

(71)出願人　596176389
トランスタール
フランス国、92100・ブローニュ、リユ・ドウ・ラ・フエルム、14

(72)発明者　エリザベト・ルブランス
フランス国、75014・パリ、リユ・ダレジア、31・ビス

(72)発明者　ロバート・カロツソ
アメリカ合衆国、マサチユーセツツ・01824、チエルムスフオード、ウオーレン・アベニユー、18

(74)代理人　弁理士　川口　義雄　（外３名）

最終頁に続く

(54)【発明の名称】　ソフトウェア工学で使われるデータモデルの取り扱い方法

(57)【要約】
【課題】　ツール間の移植性に優れたデータモデルの取り扱い方法を提供する。
【解決手段】　複数のツール（Ｔａ、Ｔｂ、Ｔｃ、Ｔｄ）間でのモデルの転送、及び、あるツールのフォーマットを中間フォーマット（ＮＩＭ）を経由して別のフォーマットに変換し得るブリッジ（Ｂａ、Ｂｂ、Ｂｃ、Ｂｄ）を使用して、データモデルを変換則モデルに適用するためのデータモデルの取り扱い方法において、先ず実行言語をもとにしてオブジェクト指向と呼ばれるアプローチによりブリッジを作成する。このブリッジは、連続して起動される複数のステップにおいて一律または条件的にデータモデルの変換を行うことができる実行可能命令であり、各ステップは、コードが再使用可能なクラスのアセンブリをもとにして開発される実行可能命令であり、プロシジャを起動する際、単に、対応するプロシジャの名前およびパラメータを与えるだけでよい。

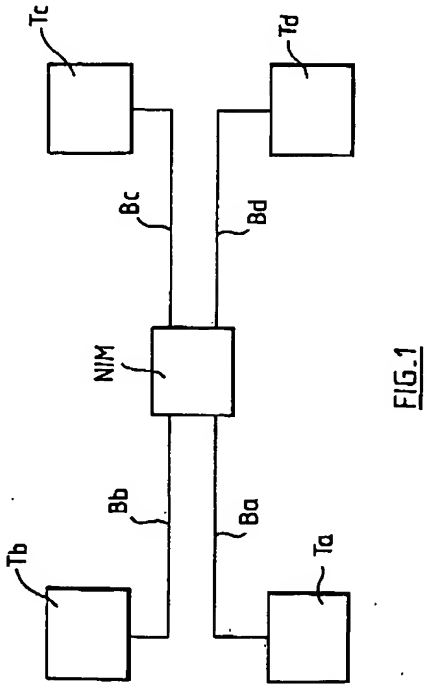

FIG.1

1

【特許請求の範囲】

【請求項1】　複数のソフトウェア工学ツール間でのモデルの転送、及び、あるツールのフォーマットを中立かつ単一の中間フォーマットを経由して別のフォーマットに変換し得るブリッジを使用して、データモデルを変換則モデルに適用するためのデータモデルの取り扱い方法であって、実行された言語をもとにしてオブジェクト指向と呼ばれるアプローチによりブリッジを作成し、該ブリッジが、ステップのダイナミックチェイニングロジックを記述する制御言語により、連続して起動される複数のステップにおいて一律または条件的にデータモデルの変換を行うことができる実行可能命令であり、チェイニングがパラメータ化され名前の付けられたプロシジャであり、各ステップが、コードが再使用可能なクラスのアセンブリをもとにして開発される実行可能命令であり、プロシジャを起動する際、単に、対応するプロシジャの名前およびパラメータを与えるだけでよいことを特徴とするデータモデルの取り扱い方法。

【請求項2】　ブリッジの開発のために、モデル化オブジェクトのダイナミックアロケーションを使用することにより、リファレンシャルまたは外部ファイル内のメモリに保存されているモデルおよびメタモデルを表すことができる再使用可能構成要素が作られ、このクラスのオブジェクトがメモリ構造全体を統合し、メモリにロードされたモデルのデータを扱うための機能を供給することを特徴とする請求項1に記載のデータモデルの取り扱い方法。

【請求項3】　ブリッジの開発のために、ツールエキスポートファイルの種々のフォーマットを符号化し解読し得る再利用可能なクラスがつくられ、このクラスはツールエキスポートのレコードをカプセル化し、インポートファイルのレコードフォーマットを記述するために使われる一方、初期化時には、メモリに保存されたモデルおよびメタモデルの表現クラスを使用して、テキストファイルからツールのメタモデルが読み込まれると同時に、ブリッジのコードによりツールエキスポートファイルを読み、レコードに固有のフィールドにアクセスし、メモリに保存されたモデルおよびメタモデルの表現クラスの機能を使用することによりオブジェクトおよびその属性をつくることができることを特徴とする請求項1または2に記載のデータモデルの取り扱い方法。

【請求項4】　ブリッジの開発のために、C型言語のキャラクタストリングとの適合性を保ちつつ、割り当て、連結、サブストリングおよび動的メモリ再割り当て機能をともなう比較演算子の引き受けが可能なキャラクタストリングの取り扱い再利用可能なクラスがつくられることを特徴とする請求項1から3のいずれか一項に記載のデータモデルの取り扱い方法。

【発明の詳細な説明】

【0001】

2

【発明の属する技術分野】本発明は、複数のソフトウェア工学ツール間でモデルを転送するためのデータモデルの取り扱い方法、ならびにあるツールのフォーマットを中立かつ単一の中間フォーマットを経由して別のフォーマットに変換することができるブリッジを使用して行われる、変換則のモデルへの適用に関する。

【0002】

【従来の技術】一般的に、ユーザまたは設計者の環境内で使われる異種開発ツール間でアプリケーションの開発に関する情報をデータモデルの形態で交換できるようにするため、ソフトウェアツール間にインタオペラビリティを求めることは、ユーザまたは設計者にとって共通かつ根本的な要求である。特にデータモデルの転送は、例えば、ソフトウェア工学ツール間、または既存のソフトウェア工学ツールとクライアント／サーバモデルの新しい開発ツールとの間でのアプリケーションのマイグレーション、あるいは例えば（当業者が「レポジトリ」と呼ぶ）リファレンシャル内でのデータモデルの形態の開発情報の管理を行えるようにするためには貴重な支援となり得る。このようなツールは次第に一般的に使用されるようになってきていることから、情報モデルの設計または分析を行うことができる市販の単数または複数のソフトウェア工学ツールのエキスポートファイルの管理、回収、または転送を行おうとする希望が出されることがある。しかしながら現状では種々の欠点が存在する。例えば各ツールは独自の内部形式ならびに個別または特有のオブジェクト表現方法、情報保存方法等を有するので、二つのツール間ではやりとりが行えず、あるいは情報を直接交換することができない。

【0003】これらの欠点を解消するために最近まで使われていた解決方法は、各ツールおよびツールからツールについて、当業者が「ブリッジ」と呼ぶもの、すなわちあるツールのフォーマットを別のツールのフォーマットに変換することを役割とするインタフェースを開発することであった。その結果、ツール間で対話が行えるようにするために、やりとりを行おうとする相手となるツールの数と同数のブリッジを各ツールについて開発しなければならなかったとともに、新しいツールが出現すると、この新しいツールとの対話ができるようにするため各ツールについて固有のブリッジを開発する必要があった。この解決方法は明らかに、可能なツールの組み合わせと同じ数のブリッジを作る必要があるという大きな欠点を有するため、開発のコストおよび時間からすぐ実現不可能になる。例えば、4個、5個または6個のツール間で相互に対話を行うという、比較的少ないツール数を想定する場合でも、必然的にそれぞれ6個、10個、15個のブリッジを開発しなければならない。

【0004】この欠点を少なくとも部分的に解消するため、やはり固有ではあるが、交換のコアとなる中立的なピボット情報モデルをつくることにより、あるツールの

フォーマットを中立かつ単一の共通中間フォーマットを経由して別のツールのフォーマットに変換することができるようにしたブリッジを開発するということからなる解決方法が考案された。全てのブリッジがこの交換コアを通過するこのピボット情報モデルを使用する中立的表現という考え方により、ブリッジを著しく減らすことが可能であり、しかもやりとりを行うべきツール数が多ければ多いほどそのことが言える。従って前記のように4個、5個または6個のツール間で相互に対話を行うとする場合、それぞれ4個、5個または6個のブリッジを開発するだけでよい、すなわち一つのツールあたり一つのブリッジにより前記交換コアにアクセスし、前記ツールのフォーマットを共有中間フォーマットに変換する。この種の解決方法の場合、すぐれた移植性を確保するため、この機能を果たすための固有言語が開発され書き込まれインタープリットされる、コンパクトで管理が簡単で機能的な調節がなされたプロプリエタリアプローチが採用された。事実、唯一つのプログラムをインタープリットするだけでよいので、この閉じた世界では移植はかなり顕著に簡単になる。この解決方法は有利ではあるが、欠点も有する。第一の欠点は、インタープリットされる言語が高速ではなく、インタープリターは前記言語を分析してから記載した動作を実行するという事実に本質的に関わる。このようなプロプリエタリ言語を使用する際の第二の大きな欠点は、その言語では再使用可能要素の概念を利用することができないことである。実際、新しいブリッジを開発しようとする場合、この固有言語で全コードを再度書き込む必要がある。さらにこのような言語は通常、開発環境が非常に悪い（再利用可能機能のライブラリがない、デバッガツールがない等）。

【0005】

【発明が解決しようとする課題】本発明は、先行技術による既知の種々の方法の種々の欠点を解消することを目的とし、使用する言語を適切に選択することにより高い柔軟性をもたらし、簡単に拡張が可能であって、かつ高いレベルの移植性および性能ならびに低いコストを実現しつつ再使用可能要素の概念を使用することがデータモデルの取り扱い方法を提供する。

【0006】

【課題を解決するための手段】この点に関し、前文において言及したデータモデルの取り扱い方法は、実行された言語をもとにしてオブジェクト指向と呼ばれるアプローチによりブリッジの作成が行われ、ブリッジが、ステップのダイナミックチェイニングロジックを記述する制御言語により、連続して起動される複数のステップにおいて一律または条件的にデータモデルの変換を行うことができる実行可能命令であり、チェイニングがパラメータ化され名前の付けられたプロシジャであり、各ステップが、コードが再使用可能なクラスのアセンブリをもとにして開発される実行可能命令であり、プロシジャを起動する際、単に、対応するプロシジャの名前およびパラメータを与えるだけでよいことを特徴とする。

【0007】このように、個別的に開発されたプロプリエタリ言語はすぐれた移植性をもたらすものであり、必要であると痛感される機能上の厳密な調整に好適であるとみなされているという好ましくない先入観はあるものの、オブジェクト指向のアプローチを採用することにより実行言語を使用するよう決定された。従って、この先入観すなわち従来の先行アプローチとは異なり、例えばC++言語など、より柔軟で、好ましくは直接コンパイルされ、メモリの管理がより細密で効果的な従来の標準言語を使用するよう決定された。また、このような言語は非常にすぐれた抽象化レベルを有し、適切なライブラリとともに使用することにより、効果および速度を向上させることができる一方、同一の構造を異なるコンテクストで使用することができることから有利な汎用性を得ることができ、従って再使用係数およびカウンタビリティ係数がより良好になる。別の利点は、この種の言語は対応するデバッガを使用して簡単にデバッグが行えることであり、それによりブリッジの開発時間をさらに少なくすることができる。最後に、このように、非常に多くの製品およびユーザが存在する標準言語を選択することにより、高品質な環境が保証される。

【0008】このようにして、あるツールから別のツール、またはあるモデルから別のモデルへの転送を行うためには、ブリッジのコードを実行できるように設計された複数の構成要素が存在し、実際には各ブリッジは、再使用可能クラスのライブラリを使用して作ったオブジェクト指向プログラムとして開発される。ブリッジは、例えば、コードすなわちインポートファイル内のシンタックス要素の認識によるファイルの物理的復号段階と、次いで、中立情報モデル内のシンタックス要素の対応が発見または生成される変換の符号化段階と、場合によっては完全性確認段階というように複数の段階で作られるので、これらクラスは種々のブリッジをつくるのに使用されることになる。これら各段階は再使用可能モジュールを構成し、モジュールのアセンブリは、プログラムとしてではなく適用する一連のプロシジャとして動的に連鎖することができる。この目的のため、操作定義ファイル（当業者からは「オペレーションデフィニションファイル」と呼ばれる）を使用して、動的に連鎖すべきモジュールのリストが記述される一方、タスクマネージャは、再使用可能プロシジャを含む実行可能命令の中から、実行可能命令を実行するための前記モジュールを探し、前記ブリッジの適合のためのコンフィギュレーションパラメータを考慮してブリッジをつくる。これら情報は全て、例えばディスクなどの常駐空間内に保存することができる一方、中間結果は一時的にメモリに保管される。さらに、適当なプロシジャおよび／または実行可能命令が同時に複数存在するときには、タスクマネージャによ

り最適な実行順序を選択することもできる。例えば、実行順序を種々のプロシジャの複数のコピーに配設することができる。その場合、タスクマネージャはモジュールのシーケンス、ならびに実行可能命令の負荷を最小限にするプロシジャクラスを自動的に選択するので、これにより実行時間を最適化することができる。

【０００９】従って本データモデル取り扱い方法により、ブリッジを短期間で開発することができる再使用可能構成要素が起動される。注目すべきは、ブリッジの開発のために、モデル化オブジェクトのダイナミックアロケーションを使用することにより、リファレンシャルまたは外部ファイル内のメモリに保存されているモデルおよびメタモデルを表すことができる再使用可能クラスが作られることであり、このクラスのオブジェクトはメモリ構造全体を内蔵し、メモリにロードされたモデルのデータを扱うための機能を供給する。

【００１０】一般的にデータは種々の形態、種々のフォーマットで、種々の場所に保存することができる。好ましくは、取り扱いを簡単にするために、ここでは以下の機構を特別扱いにする。

【００１１】－　ツールによりプロプリエタリフォーマットにおいて作られたユーザデータファイルであるツールエキスポートファイルを読み、書き、分析することができる機構

－　例えばＣ＋＋オブジェクトなどのオブジェクトに対応し、メモリ構造アセンブリをそれぞれカプセル化または内蔵し、ツールエキスポートファイルおよび／またはリファレンシャルをもとにしてロードされたデータモデルを取り扱うための機能を供給する機構であって、これらオブジェクトの実施およびこれらオブジェクトを取り扱う機能が再使用可能クラスを構成する機構。あるモデルのデータの編成は、これらクラスの生成時にロードしたメタモデルにもとづき、メタモデルは、メタモデルファイルと呼ばれるテキストファイルからの実行時にロードされることが好ましい。このようなクラスはツールのメタモデルあるいは中央情報モデルのメタモデルによりロードされることができ、これら二種類のオブジェクトはブリッジの実行中に使用される。

【００１２】－　中立情報モデルのデータとの対応が行われたデータを含むテキストファイルである中立データファイルと呼ばれる機構であって、種々の実行可能命令間におけるデータの転送用またはディスクへの永久保存用として使われる機構

－　中立情報モデルを示す略図を使用することによりユーザのデータを保存することができ、リファレンシャルのツール（例えば、バージョン、要求の管理等）により管理ができるリファレンシャル

このように、ブリッジによりデータをある場所またはある保存機構から別の場所または保存機構におよび／またはある形態から別の形態に移動および操作することができる。それにはインポートブリッジ、エキスポートブリッジ、または転送ブリッジという三つのブリッジ例が与えられる。インポートブリッジをつくることにより、ツールのエキスポートファイルからデータを抽出してこれをリファレンシャル内に入れることができる一方、エキスポートブリッジをつくることにより、リファレンシャルからデータを抽出してツールのエキスポートファイルをつくることができる。転送ブリッジをつくることにより、リファレンシャル内の保存を経ずにあるツールのエキスポートファイルから別のファイルに直接データを回収することができる。もちろん、データを移動したり符号化することを必ずしも目的とせず、ある所与の時点でメモリにロードされるデータを処理することができる別の種類の作業も存在する。従ってこれら作業は全て、データをメモリにロードできること、およびデータを永久的に保存することによりデータの取り扱いおよび書き込みができることを必要とし、これら作業は、ここに請求項として示す再使用可能クラスおよび機能が存在することにより実現可能である。作業によっては多重オブジェクトの生成を必要とし、結果として種々のオブジェクト間で変換または符号化がなされるものもある。例えばこれらデータがツールエキスポートファイルから抽出される時には、実行可能命令により、このファイルを構文的に分析し、ツールのメタモデルに対応するような構造を有するエキスポートファイルを含むオブジェクトをつくることができる。次に、中立情報モデルに対応するオブジェクトへのこのオブジェクトの変換を行う中立情報モデルにより、ツールの固有データが一般データに変換される。このようにして保存されたデータは処理することができ、場合によっては中立データファイルに保存することが可能であり（データを別の実行可能命令に送る場合）、あるいはリファレンシャル内に保存することができる。

【００１３】この結果、ブリッジによりモデルのオブジェクトが動的にメモリに割り当てられるので、データモデルおよびメモリに保存されたメタモデルの定義にアクセスすることができる。メタモデルのロード後は、メタモデルのゲージを使用することによりモデルがつくられる。モデルの集合体はメタモデルの集合体の構造と同様の構造を有するので、これにより、オブジェクト間、および属性の定義とそのインスタンスとの間で素早く対応関係をとることができる。

【００１４】ブリッジを開発するために、ツールエキスポートファイルの種々のフォーマットを符号化し解読することが再利用可能クラスがつくられることになるのも特徴である。このクラスはツールエキスポートのレコードをカプセル化し、インポートファイルのレコードフォーマットを記述するために使われる一方、初期化時には、メモリに保存されたモデルおよびメタモデルの表現クラスを使用して、テキストファイルからツールのメタ

モデルが読み込まれると同時に、ブリッジのコードによりツールエキスポートファイルを読み、レコードに固有のフィールドにアクセスし、メモリに保存されたモデルおよびメタモデルの表現クラスの機能を使用することによりオブジェクトおよびその属性をつくることができる。

【００１５】事実、Ｃ＋＋言語も含めプログラミング言語では、複数の可変フィールドを含む構造を直接取り扱うことができない。ここで提案する解決方法は、一連のフィールド記述子の形態のレコード宣言を使用することである。プログラマは、固定および可変フィールドの大きさ、値として定数および区切り記号の値をもつレコードのコードを定義する。処理コードにおいては、プログラマはまず、レコードをその記述子に対応させることができる機能を呼び出し、次にレコードのフィールドにアクセスする。また、エキスポートファイルのレコードをつくるために、プログラマはモデルのデータをもとにしてこのレコード固有のフィールドに書き込み、次にレコード全体を書き込む。レコードの記述には、各フィールドについて、実行時の情報の名前、種類および大きさが含まれる。記述情報およびデータはデバッギングのために表示することができる。

【００１６】このようにして特許請求するこのクラスによって、フィールドからのデータをレコード内に供給し、レコードのフィールド内にデータを固定し、抽出のための完全なレコードを供給することにより、レコード記述子をつくるためのマクロ、およびレジスタをレコードの記述と対応させるための機能プロトタイプを定義することができる。

【００１７】このように、この再利用可能クラスにより、ブリッジの構成要素を使用して読み書きを行う種々のツールエキスポートファイルの種々の処理を有利に行うことができる。

【００１８】最後に、同じく注目すべきは、ブリッジを開発するために、Ｃ型言語のキャラクタストリングとの適合性を保ちつつ、割り当て、連結、サブストリングおよび動的メモリ再割り当て機能をともなう比較演算子の引き受けが可能なキャラクタストリングの取り扱い再利用可能クラスがつくられることである。

【００１９】このように、この再利用可能クラスにより、Ｃ言語においてあらかじめ定義されているデータ種の挙動と同様の挙動を提供するキャラクタストリングのある種の固有データ種を提案することにより、実用的な技術的解決方法が提供される。また、この再利用可能クラスは大きな特徴を有する。なぜなら、このクラスにより、一般的にメモリブロックの欠落にともなう情報の欠落が一切禁止されるからである。

【００２０】

【発明の実施の形態】図１は、本発明による、ソフトウェア工学ツールＴａ、Ｔｂ、Ｔｃ、Ｔｄ、．．．、の集合体の中での有利なインタオペラビリティーを示す。前記ツールは、例えばアプリケーションの開発に関する情報をデータモデルの形態で交換する。ツールＴａ、Ｔｂ、Ｔｃ、Ｔｄは、それぞれ独自の環境下で独自の辞書とともに使用される異種開発ツールである。これら四つのツール間のデータモデルの変換は、ツールＴａ、Ｔｂ、Ｔｃ、Ｔｄのうちの一つの独自のフォーマットを中立かつ単一の中間フォーマット、中立情報モデルＮＩＭ、を経由してツールのうちの別のツールの独自のフォーマットに変換することができるＢａ、Ｂｂ、Ｂｃ、Ｂｄなどのブリッジを使用して行われる。より正確には、交換される情報は、結果的にピボットモデルとなるモデルＮＩＭとの間でデータを送受信する前記ツールのエキスポートファイルを経由する。中立情報モデルＮＩＭをインスタンスするデータはリファレンシャルと呼ばれるオブジェクトデータベース内に保存される。

【００２１】図２には、本発明を実施することができる種々の構成要素が示されるアーキテクチャの一例を示す。種々の構成要素とは、インタフェース、コントローラ、保存機構、コンフィギュレーションファイル、コントローラ標準シーケンス（本発明により実行される言語においてはスクリプト）、ブリッジ、およびリファレンシャルである。この環境は、開発のツールボックスであるとみなすことができる。このツールボックスの作用は、プログラマインタフェースクラスとも呼ばれるコントロールクラスＣＣによってモニタされ、このクラスにより、環境をデータ転送用に編成すると同時に、適切なプロシジャを呼び出すことができる。ユーザグラフィックインタフェースＧＵＩ、コマンドラインインタフェースＣＬＩ、あるいはその他のインタフェースなど、ワークステーションＷＳの画面上での表示用のインタフェースの開発者は、専らコントロールクラスＣＣを使用する。このコントロールクラスＣＣは、ブリッジを呼び出しあらゆるコマンドを起動することができる適用インタフェース（ＡＰＩ）機能を含む。ここで、ブリッジとは、データをある場所から別の場所に移動すること、およびある形態から別の形態に操作することができる特有の機構であること、ならびに特に、インポート、エキスポート、および転送の三種類のブリッジがあることを再度記しておく。コントロールクラスＣＣはコンフィギュレーションファイルＣＦおよびコントロール標準シーケンスＣＳ（スクリプト）によりパラメータ化される。データは種々の形態およびフォーマット、かつ異なる場所で、メモリにロードすることができる。それぞれが独自の有用性をもちかつ独自の組み合わせサービス群を有する四つの保存機構が使われる。すなわち、ツールエキスポートファイルＴＥＦ、中立データファイルＮＤＦ、リファレンシャルＲ、およびモデルオブジェクトの動的割り当て機構Ｄである。実際には、ツールエキスポートファイルＴＥＦは、外部ツールがもつフォーマット内に前

記ツールによってつくられるユーザデータファイルであり、このファイルＴＥＦは本発明の手段により、読み出し、書き込み、および分析を行うことができる。中立データファイルＮＤＦは、インポートされ当初ツールのメタモデルに従った構造をもつモデルの中立情報メタモデルによるモデリングの結果を含むテキストファイルである。このファイルＮＤＦは、モデルのこの中立表現をＡＳＣＩＩで示したものである。中立データファイルＮＤＦの読み出しおよび書き込みサービスは本発明に従い供給される。中立データファイルにより、データの欠落を防ぎつつ、ある実行可能命令のＣ＋＋オブジェクトに関するデータを別の実行可能命令のＣ＋＋オブジェクトに転送することができる。リファレンシャルＲは、中立情報モデルを使用することができる略表を含む。ユーザのデータは、中立情報モデルのフォーマットでリファレンシャル内に保存し、次に、バージョン、要求などの管理を含むリファレンシャルのツールにより管理および操作することができる一方、データをリファレンシャル内に保存し、あるいはリファレンシャルからデータを抽出するためにサービスが供給される。中立データファイルＮＤＦは、オブジェクトとその内容としてリファレンシャル内に保存することもできる。モデルオブジェクトの動的割り当て機構Ｄも本発明の特長であり、完全に包括的に、かつ実行時にロードされるメタモデルのデータによって指定される方法によりデータモデルを表すのに使用される一式のＣ＋＋クラスを供給する。メタモデルは、メタモデルファイルと呼ばれるテキストファイルからロードされ、Ｃ＋＋オブジェクトとして保存される。Ｃ＋＋オブジェクトとして次に保存されるユーザデータは、組み合わされるメタモデル内で対応関係がとられる。なお、中立情報モデルは、全てのツールメタモデルの概念の大部分を獲得できるよう充分に完備されるようにしたメタモデルである。中立情報モデルのメタモデルとともにロードされるＣ＋＋オブジェクトは、実際には、そこから別の保存フォーマットをつくることができる中央または中心保存フォーマットである。オブジェクトの動的割り当て機構Ｄは、本発明の実施に必要なクラスを全て含むと同時に、全てのメタモデルＭＴａ、ＭＴｂ、．．．、ＮＩＭＭＴを含む。

【００２２】四つの保存機構すなわち、ファイルＴＥＦ、ファイルＮＤＦ、リファレンシャルＲ、および機構Ｄは、ブリッジおよび操作機構ＯＢＭと直接の関係にあり、この機構もコントロールクラスＣＣとの関係にある。

【００２３】前記に説明し、さらに図２、特にブリッジおよび操作機構ＯＢＭとの関係において説明したように、あるツールから別のツール、あるいはあるフォーマットから別のフォーマットへの転送を行うために、ブリッジのコードを実行することができるように設計された複数の構成要素が存在するが、実際には各ブリッジは、再利用可能クラスのライブラリを使用して作成したオブジェクト指向プログラムとして開発された作業である。ブリッジは、例えば、コードすなわちインポートファイル内のシンタックス要素の認識によるファイルの物理的解読段階と、次いで、中立情報モデル内のシンタックス要素の対応が発見または生成される変換の符号化段階と、場合によっては完全性確認段階（機構ＩＣ）というように複数の段階で作られるので、これらクラスは種々のブリッジＢａ、Ｂｂ、Ｂｃをつくるのに使用される。これら各段階は再使用可能モジュールを構成し、モジュールのアセンブリは、プログラムとしてではなく一連のプロシジャに従って動的に連鎖することができる。この目的のため、操作定義ファイルＯＤＦ（当業者からは「オペレーションデフィニションファイル」とも呼ばれる）を使用して、動的に連鎖すべきモジュールのリストが記述される一方、このファイルは、再使用可能プロシジャを含む実行可能命令の中から、実行可能命令を実行するためのモジュールを探し、前記に説明したように実行時間の有利な短縮をはかりつつ、ブリッジの適合のためのコンフィギュレーションパラメータを考慮してブリッジをつくるタスクマネージャＴＭによって利用される。これらパラメータは全て、常駐空間（ＣＳ）内に保存することができる一方、中間結果は一時的にメモリに保管される。従って、本データモデル取り扱い方法により、ブリッジを短期間に開発することができる再利用可能構成要素が起動される。ブリッジ以外の作業は、例えば完全性確認ＩＣあるいは名前変換サービスＮＣＳなど、データが必ずしも移動または変換されるとは限らないスタンドアローン作業として実行することができる。

【００２４】本発明の趣旨をよりわかりやすくするため、いくつかの定義および本発明にしたがい請求する種々のクラスの可能であるが非限定的な実施についての説明を以下に示す。

【００２５】まず、アプリケーションの構成要素は、本発明の環境のユーザにインタフェースを提供するプログラムであることを示して、「スクリプト」の編成およびコンフィギュレーションに関するクラスの定義を示す。インタフェースはどのような種類であってもよい。アプリケーションはコントロールクラス（ＴｓＣｏｎｔｒｏｌ　Ｃｌａｓｓ）のオブジェクトをユーザ名にインスタンスし、次に、このユーザに許可された作業を知るための機能を呼び出す。ユーザは、これらの作業の中から選択し、前記作業を実行すべき方法についての情報を供給するか、これら命令および情報を含むあらかじめ定義されたコントロール「スクリプト」を指定することができる。アプリケーションによって可能であれば、あらかじめ定義されたこれら「スクリプト」を作成するのにも、コントロールクラス（ＴｓＣｏｎｔｒｏｌ　Ｃｌａｓｓ）を使用することができる。

【００２６】コントロールクラス（ＴｓＣｏｎｔｒｏｌ

Ｃｌａｓｓ）は、本発明の環境下で作業を構成し開始するために使われる。アプリケーションは、コントロール「スクリプト」を読み書きし、可能な作業のリストを作成するための機能を使用する。「ｓｅｔ」および「ｇｅｔ」機能を使用することによりキーワード値を定義することができ、「ｄｏ」機能を使用することにより動作を命令することができる。機能が適用されるキーワード＊または作業をユーザが指定する時、これら機能のうちのいくつかは総体的であるが、大部分の機能は、キーワードまたは既知の作業に固有である。後者の機能は、適切なインタフェースおよびアプリケーションの構成要素のコードとともに使用されるようになっている。

【００２７】

```
TsControlクラス
    {
    TsSysConfig    sysconf;    //コンフィギュレーションファイルオブジェ
クト
    TsMessageLib    msglib;    //メッセージライブラリオブジェクト
    TsOperCtlScript            //コントロールスクリプト
        *operctl;          //オブジェクトポインタ
    TsStr    opername;         //作業名
    Singly_linked_list(TsKwdInfoPtr)      //キーワードリスト
           keywords;           //
    Doubly_linked_list(TsStr)      //使用可能ツールリスト
           *tools_list;        //
    int       error;             //エラー状態値
//共通で既知のキーワードの記憶値
    TsTokens    importfiles;      //インポートファイルリスト
    TsTokens    exportfiles;      //エキスポートファイルリスト
    TsStr      srcpctepath;      //リファレンシャルへのアクセス名（ソー
ス）
    TsStr    tarpctepath;      //リファレンシャルへのアクセス名（ターゲ
ット）
    TsStr    srctoolname;      //ソースツール名
    TsStr    tartoolname;      //ターゲットツール名
    TsStr    logfilepath;      //ジャーナルファイルアクセス名
    TsBoolean    autonaming;      //標準名前フラグ
    TsBoolean    autochecking;    //完全性確認フラグ
//内部機能
    Doubly_linked_list(TsStr)*    //
    GetGroupPrompts(              //グループ名前読み出し
    Singly_linked_list(TsGroupInfoPtr)    //情報のグループ化および請求者
への戻し
              *group_list  );//vers le demandeur
    TsKwdInfo *FindKeyword(      //キーワードの定義の検索
       const TsStr& keyword)const;      //所与のキーワードのためのオブ
ジェクト
    void AddKeywords(const      //リストにキーワードを追加
    Singly_linked_list(TsKwdInfoPtr)&      //キーワードリスト
              kwd_list      );//
//作成および分解機能
    TsControl(                  //      コンストラクタ
    const TsStr&  sys_config_path,      //コンフィギュレーションファイル
のパス  TsStr&
    const TsStr&  msg_lib_path      //メッセージライブラリのパス
    );              //
```

```
    13
    virtual～TsControl();      //デストラクタ
//コントロール「スクリプト」のロードおよび作成とその実行
    int LoadNewOperCtlScript(      //コントロール「スクリプト」のロード
      const TsStr& ctl_script_path  );//
    int WriteOperCtlScript();      //コントロール「スクリプト」への数値
の書き込み
    int CreateOperCtlScript(      //コントロール「スクリプト」の作成
      const TsStr& ctl_script_path  );//
    int RunOperCtlScript();      //コントロール「スクリプト」の実行
//作業名設定および読み出し機能
    void SetOperationName(      //作業名の設定
          const TsStr& opname  );//
    const TsStr& GetOperationName() const;      //作業名の読み出し
//既知のキーワードの値の設定および読み出し機能
    void SetTargetPctePath(      //リファレンシャルパス（ターゲット）の
設定
          const TsStr& tpath  );//
    const TsStr& GetTargetPctePath() const;      //リファレンシャルパス
（ターゲット）の読み出し
    void SetSourcePctePath(      //リファレンシャルパス（ソース）の設定
          const TsStr& spath   );//
    const TsStr& GetSourcePctePath() const;      //リファレンシャルパス
（ソース）の読み出し
    void SetTargetToolName(      //ターゲットツール名の設定
          const TsStr& tname  );//
    const TsStr& GetTargetToolName() const      //ターゲットツール名の読
み出し
    void SetSourceToolName(      //ターゲットツール名の設定
          const TsStr& sname  );//
    const TsStr& GetSourceToolName() const;      //ソースツール名の読み
出し
    void SetLogFilePath(      //ジャーナルファイルパスの設定
          const TsStr& lpath  );//
    const TsStr& GetLogFilePath() const;      //ジャーナルファイルパスの
読み出し
    void SetAutoNamingOn();      //名前の標準化ＯＮ
    void SetAutoNamingOff();      //名前の標準化ＯＦＦ
    void SetAutoNamingFlag(      //名前フラグの値の設定
          const TsStr& name_flag  );      //フラグ文字ストリング
    int GetAutoNamingState();      //名前フラグ状態読み出し
    const char *const GetAutoNamingFlag();      //名前フラグ読み出し
    void SetAutoCheckingOn();      //完全性確認指示
    void SetAutoCheckingOff();      //完全性確認指示せず
    void SetAutoCheckingFlag(      //完全性確認フラグ値設定
          const TsStr& check_flag  ); //文字ストリングフラグ
    int GetAutoCheckingState();      //完全性確認フラグ状態読み出し
    const char *const GetAutoCheckingFlag();      //完全性確認フラグ値読
み出し
    void SetImportFiles(      //インポートファイル名追加
          const TsStr& ifiles  );      //インポートファイル名ストリング
```

```
    void AddImportFile(      //インポートファイル名追加
         const TsStr& ifile  );      //インポートファイル名
    void RemoveImportFile(      //インポートファイル名削除
         const TsStr& ifile  );      インポートファイル名
    int IsImportFileOnList(      //リスト上での名前の有無のチェック
         const TsStr& ifile  );      //インポートファイル名
    const TsStr& GetImportFilesStr();      //インポートファイル名ストリ
ング読み出し
    const Doubly_linked_list(TsStr)&      //
         GetImportFilesList();      //インポートファイル名リスト読み出
し
    void SetExportFiles(      //エキスポートファイル名追加
         const TsStr& efiles  );      //エキスポートファイル名ストリン
グ
    void AddExportFile(      //エキスポートファイル名追加
         const TsStr& efile  );      //エキスポートファイル名
    void RemoveExportFile(      //エキスポートファイル名削除
         const TsStr& efile  );      //エキスポートファイル名
    int IsExportFileOnList(      //リスト上での名前の有無のチェック
         const TsStr& efile  );      //エキスポートファイル名
    const TsStr& GetExportFilesStr();      //エキスポートファイル名スト
リング読み出し
    const Doubly_linked_list(TsStr)&      //
         GetExportFilesList();      //エキスポートファイル名リスト読み
出し
    const Doubly_linked_list(TsStr)*      //
    ListTools();      //使用可能ツール名読み出し
//一般キーワード機能
    void SetKeywordValue(      //リストへのキーワードの追加
         const TsStr& name,      //キーワード名ストリング
         const TsStr& value    );//キーワード値ストリング
    const TsStr& GetKeywordValue(const TsStr& keyword)const;      //あ
るキーワード名についてキーワード値検索
    void RemoveKeyword(const TsStr& keyword);      //あるキーワードにつ
いてキーワードオブジェクト定義検索
    Doubly_linked_list(TsStr)*  GetGroups(const  TsKwdExpr& exp);      /
/グループのミーティングの基準の名前検索
    Doubly_linked_list(TsStr)*  GetGroups(const  TsKwdInfo& inf);      /
/グループのミーティングの基準の名前検索
//個別作業（インポート、エキスポート等）の実行機能
    void DoImport();      //インポート作業実行
    void DoExport();      //エキスポート作業実行
    void DoTransfer();      //転送作業実行
//その他のクラスの機能
    operator int()const;      //整数記録
    void Print();      //オブジェクト内の値のプリント
};                    //ＴｓＣｏｎｔｒｏｌクラス終了
```

ＳｙｓＣｏｎｆｉｇクラスは、本発明の環境内のコンフ　＊のアクセス名を設定することができる。このクラスは二
ィギュレーションファイルの個別クラスである。インス　つの方法で使用することができる。
タンスすることによりコンフィギュレーションファイル＊50　【００２８】第一の方法によれば、インストレーション

17

時、特殊フラグおよびコンフィギュレーションファイルのアクセス名とともに呼び出される実行可能命令によりこのクラスがインスタンスされ、ファイルの最後に（スタティックテーブルから）実行可能命令のカレントコンフィギュレーションテーブルを呼び出すためのスタティック機能"SysConfig::AppendLocalConfiguration()"が呼び出される。ファイル内に前に存在していた現在の実行可能命令に関する情報はなくなる。

【００２９】第二の方法によれば、ＳｙｓＣｏｎｆｉｇクラスは、コンフィギュレーションファイルの記録および

18

＊び分析、ならびに実行可能命令、グループ、方法等のデータ構造の作成の"SysConfig::LoadFile()"機能を有し、この機能は、現在のインストレーション内に存在するものを表す。LoadFile()機能も、現在の実行可能命令のコンフィギュレーションがコンフィギュレーションファイル内にあるものに対応していることを確認する。一連の"SysConfig::Find...()"機能により、コンフィギュレーション情報の回収のためにシステムの構造に素早くアクセスすることができる。

【００３０】

```
ＳｙｓＣｏｎｆｉｇクラス
  {
  TsStr pathname;      //コンフィギュレーションファイルへのアクセス名
  TsFileReader *input;      //コンフィギュレーション読み出しのためのオブジェクト
  TsFileWriter *output;      //コンフィギュレーション書き込みのためのオブジェクト
  Singly_linked_ring(TsExecInfo*)exec_list;      //コンフィギュレーションデータ構造
  TsExecInfo  *curr_exec;      //実行の現在情報へのポインタ
  TsGroupInfo  *curr_group;      //現在グループへのポインタ
  TsMethodInfo  *curr_method;      //現在方法へのポインタ
  SysConfig(const TsStr& path);      //コンストラクタ
  ～SysConfig();      //デストラクタ
  int LoadFile();      //ファイルの読み出しおよび分析
  TsExecInfo *Find Exec(const TsStr& name,      //指名実行可能命令の検索
    const TsStr& location);  //
  TsGroupInfo *FindGroup(const TsStr& name);      //指名グループの検索
  TsMethodInfo *FindProc(const TsStr& name);      //指名方法の検索
  static int AppendLocalConfiguration();      //ファイル内への情報の書き込み
  static int DeleteLocalConfiguration();      //ファイル内の情報の削除
  }
```

ＯｐｅｒＣｔｌＳｃｒｉｐｔクラスは、本発明の環境内のコントロールスクリプトの個別クラスである。インスタンスすることにより、コントロールスクリプトのアクセス名を設定することができる。対応するキーワード定義リストとともに、実行すべき作業のリストを作成することによりコントロールスクリプトを登録し分析するために、"OperCtlScript::LoadFile()"機能が使われる。

※【００３１】"OperCtlScript::SaveFile()"機能は、コントロールスクリプトの定義およびコマンドを出力するために使われる。またこのクラスは、コマンドの実行および次の実行可能命令のコンフィギュレーションおよび呼び出しのための機能も含む。

【００３２】

※

```
ＯｐｅｒＣｔｌＳｃｒｉｐｔクラス
  {
  TsStr pathname;      //スクリプトアクセス名
  int temp;      //スクリプト一時フラグ
  TsFileReader *input;      //スクリプト読み出しのためのオブジェクト
  TsFileWriter *output;      //スクリプト書き込みのためのオブジェクト
  TsExecInfo *curr_exec;      //現在の実行可能命令
  TsGroupInfo *curr_group;      //現在のグループ
  TsMethodInfo *curr_method;      //現在の方法
```

```
Singly_Linked_List(TsMethodInfo*)commands;      //実行すべきコマンド
Singly_Linked_List(TsDef*)defines;      //コマンド用パラメータ
SysConfig *system;      //ＳｙｓＣｏｎｆｉｇオブジェクトへのポイン
タ
TsStr    next_exec;      //行うべき次の実行
TsStr    next_loc;      //次の実行の場所
OperCtlScript(const TsStr& path);      //コンストラクタ
～OperCtlScript();      //デストラクタ
int LoadFile();      //ファイルの読み出しおよび分析
int SaveFile();      //ファイルのコンフィギュレーションの書き込み
int SaveFile(const TsStr& new_path);      //新規ファイルの作成
int RunCommands();      //方法呼び出し
int PassExecution();      //新規実行可能命令の呼び出し
}
```

ＴｓＤｅｆクラスは、コントロールスクリプトのキーワ　＊【００３３】
ードおよび値を保存し回収するのに使われる。　＊

```
ＴｓＤｅｆクラス
{
TsStr keyword;      //キーワード識別子
TsStr value;      //キーワード値
TsDef(const TsStr& kwd, const TsStr& val);      //コンストラクタ
～TsDef();      デストラクタ
void SetValue(const TsStr& val);      //キーワード値設定
const TsStr& GetKeyword();      //キーワード識別子読み出し
const TsStr& GetValue();      //キーワード値読み出し
}
```

ＴｓＫｗｄＩｎｆｏクラスは、キーワードの保存および　※ーワードの現在値（最も最近の定義）を含むＴｓＤｅｆ
回収、ならびにコンフィギュレーションファイルの識別　オブジェクトへのポインタも保存する。
子の要求に使われる。このクラスのオブジェクトは、キ※　【００３４】

```
ＴｓＫｗｄＩｎｆｏクラス
{
TsStr keyword;      //キーワード識別子
int prompt_id;      //案内ストリングメッセージ
TsDef *definition;      //キーワード値オブジェクトへのポインタ
TsKwdInfo(const TsStr& kwd,      //コンストラクタ
    const TsStr& prmt);      //
～TsKwdInfo();      //デストラクタ
int SetDef(const Singly_linked_list(TsDef*)& defs);      //キーワー
ド値設定
const TsStr& GetKeyword();      //キーワード識別子の読み出し
const TsStr& GetValue();      //キーワード値の読み出し
const TsStr& GetPrompt();      //案内ストリングの読み出し
}
```

ＴｓＭｅｔｈｏｄＩｎｆｏクラスは、キーワードおよび　★る。
オプションが組み合わされた方法を識別するのに使われ★　【００３５】

```
ＴｓＭｅｔｈｏｄＩｎｆｏクラス
{
TsStr name;      //方法名
TsGroupInfo *group;      //グループオブジェクトへのポインタの復帰
Singly_linked_list(TsKwdInfo *)kwd_list;      //キーワードリスト
```

２１　　２２

```
    Singly_linked_list(TsKwdInfo *)opt_list;      //オプションリスト
    TsMethodInfo(const TsStr& nam, TsGroupInfo *grp);      //コンストラ
クタ
    ～TsMethodInfo();      //デストラクタ
    const TsStr& GetName();      //方法名読み出し
    TsGroupInfo *GetGroup();      //グループオブジェクトへのポインタの読
み出し
    void AddKeyword(TsKwdInfo *kwd);      //リストへのキーワードの追加
    void AddOption(TsKwdInfo *opt);      //リストへのオプションの追加
    TsKwdInfo *FindKey(const TsStr& key_name);      //キーワードの読み出
し
    }
```

ＴｓＧｒｏｕｐＩｎｆｏクラスは、グループおよび組み　＊【００３６】
合わされた方法を識別するのに使われる。　＊

```
    ＴｓＧｒｏｕｐＩｎｆｏクラス
    {
  TsStr name;      //グループ名
    TsExecInfo *exec;      //実行可能命令へのポインタの復帰
    Singly_linked_list(TsMethodInfo *)method_list;      //方法のリスト
    TsGroupInfo(const TsStr& nam, TsExecInfo *exc);      //コンストラク
タ
    ～TsGroupInfo();      //デストラクタ
    const TsStr& GetName();      //グループ名読み出し
    TsExecInfo *GetExec();      //実行可能命令オブジェクトへのポインタの
読み出し
    void AddProc(TsMethodInfo *method);      //リストへの方法の追加
    TsMethodInfo *FindProc(const TsStr& method_name);      //方法の読み
出し
    }
```

ＴｓＥｘｅｃＩｎｆｏクラスは、実行可能命令および組　30※【００３７】
み合わされたグループを識別するのに使われる。　※

```
    ＴｓＥｘｅｃＩｎｆｏクラス
    {
    TsStr name;  //実行可能命令の名前
    TsStr location;      //実行可能命令のアクセス名
    Singly_linked_list(TsGroupInfo *)group_list;      //グループリスト
    TsExecInfo(const TsStr& nam,      //コンストラクタ
        const TsStr& location);      //
    ～TsExecInfo();      //デストラクタ
    const TsStr& GetName();      //実行可能命令名読み出し
    const TsStr& GetLocation();      //実行可能命令パス読み出し
    void AddGroup(TsGroupInfo *group);      //リストへのグループの追加
    TsGroupInfo *FindGroup(const TsStr& group_name);//グループの読み出し
    }
```

次に、メモリに保存されているモデルとメタモデルの表現およびモデリングオブジェクトの動的割り当てに関するクラスの定義を示す。使用する機構（図２においては機構Ｄと呼ばれる）により、メモリ内にモデルのオブジェクトを動的に割り当てることによって、ブリッジの構成要素によるアクセスについて、メタモデルの定義およ★びデータのモデルをメモリに保存することができる。この機構によりメタモデルをロードすることができ、次にメタモデルのゲージを使用することによりモデルを作成することができる。このような構造により、オブジェクト、属性定義、およびそれらのインスタンスの間の対応関係を素早くとることができる。機構Ｄは、隠されたプ

ライベートクラスおよびパブリックインタフェースクラスを有する。オブジェクトにデータを保存しそれにアクセスするために、プログラムはパブリックインタフェースクラスＴｓＯＢＪおよびＴｓＬＮＫの機能を使用する。アドレシングオブジェクトは、オブジェクトまたはリンクのクラス、オブジェクトのインスタンス、属性の種類、および値のラインのリストについてのポインタの集合体のツリー内のインデックスを使用し、インタフェースがインデックス値に対応させるテキスト名として、オブジェクトまたはリンクのクラスおよび属性の種類が与えられる。機構Ｄは、データの階層化一覧への直接アクセスのためのポインタの集合体の他に、選択されたオブジェクトへの高速アクセス（二分法検索）を提供し、キーフィールドによって分類されたオブジェクトの表示を行うことができる対応するオブジェクトのキー属性の値によって分類されたインデックスの集合体を有する。

【００３８】機構Ｄの「コンストラクタ」は、属性およびオブジェクト定義ファイルを読み出すためにｒｅａｄＭｅｔａＭｏｄｅｌ機能を呼び出す。このルーチンは、オブジェクトクラスの定義のアドレスを、クラス名順に並べたオブジェクトクラスの集合体に挿入することにより、各オブジェクトの種類についてのオブジェクトクラスの定義を作成する。機構Ｄはヘリテージをサポートする。機構はさらに、各属性を、属性名順に並べた属性の集合体に挿入する。オブジェクトおよび属性がいったん定義されれば、オブジェクトクラスのインデックスおよびオブジェクトクラスの属性の各インデックスは固定され、モデルの作成用として使用する準備ができている。

【００３９】ツールはこの機構Ｄ内でモデル（ツールまたは中立情報モデルＮＩＭ）を作成し操作する。属性値を設定するためにツールが読み出されると、機構Ｄは、インデックス値との間で名前の対応関係がとられ要求によってつくられたオブジェクトが既に存在する時に最適化される以下のステップを実行する。

【００４０】－　ｃｌＩＮｄｘを読み出すための、オブジェクトクラスの表におけるオブジェクト種類の名前の検索

－　モデルのオブジェクトの表内のｃＩＮｄｘ位置において、オブジェクトのインスタンス用の要素を見つけ出すか作成し、新規インデックスｏｂｊＮｄｘを割り当てること

－　新規オブジェクトを作成し、そのアドレスをオブジェクトのインスタンスの表内に保存すること

－　ｐｒｏｐＮｄｘインデックスを見つけ出すため、属性定義表内で属性名を検索すること

－　オブジェクトの属性のインスタンスの表内のｐｒｏ＊

＊ｐＮｄｘ位置において、属性値のアドレスを見つけ出すか新規アドレスを作成し、このアドレスをその位置に保存すること

－　属性値のリストにおいてライン番号を検索し、見つかれば、値を属性値オブジェクト内に保存し、見つからなければ、新規属性値をつくりリスト内に挿入すること

－　属性値ポインタの保存場所を見つけ出すために、属性のインスタンスの表をインデックスすること

－　キー属性の一次値または二次値を保存することにより、キー値によって並べられた対応するインデックステーブル内にＯｂｊＮｄｘを挿入するか、インデックステーブル内のＯｂｊＮｄｘを更新すること

属性のインスタンスの表は、現在および以前のクラスの属性のポインタの連結である。

【００４１】ＴｓＤｙｎａｍｏ、ＴｓＯＢＪ、ＴｓＬＮＫクラスは、機構Ｄへのアクセスのためのユーザインタフェースを定義する。ＴｓＯＢＪクラスにより、機構Ｄのオブジェクト内にアドレシング機構を提供することによって、クラス、オブジェクト、属性およびラインをインデックスすることができる。ＴｓＬＮＫクラスは、機構Ｄのリンクのアドレシングを行う。機構Ｄのインタフェースはユーザにアプリケーション領域（データのモデリング）を提供し、ポインタ、構造、およびＣ＋＋メモリ割り当てを隠す。ＴｓＯＢＪ反復子は全オブジェクトクラス上、オブジェクトクラスの全てのオカレンス上、または全属性上でループする。プログラムはモデル上を走るか、オブジェクトおよび属性の一部またはサブツリーを選択することができる。ＴｓＬＮＫ反復子（ＴｓＯＢＪ反復子から派生）は、全リンク上でループする。プログラムは、オブジェクト間のリンクを横断することによりモデルを検索することができる。リンクはＴｓＯＢＪクラスからこの可能性を継承しているので、属性をもつことができる。

【００４２】ＴｓＤｙｎａｍｏクラスは、操作途中のメタモデルおよびモデルを指定するオブジェクト（Ｃ＋＋オブジェクトであって、以降ダイナモオブジェクトと呼ぶ）を有する。ＴｓＯＢＪオブジェクトはダイナモオブジェクトを指定し、モデル内の位置のインデックスを含む。ＴｓＯＢＪオブジェクトの位置は、文字ストリング内のクラスおよび属性の名前および／またはインデックス値により設定することができる。これにより、実行時の名前の多価値性をサブスクリプトの使用時の効果に組み合わせることができる。インデックス値１はデフォルトに相当する。

【００４３】

```
#include<tsdynamo.h>
TsDynamo PAC("pac","Test Model");
//PACLAN／X（ツール）の作成
```

２５

```
TsDynamo NIM("nim","Neutral Model");　　//ＮＩＭの作成
TsOBJPAC_ORB(PAC,"ORB");
while(PAC_ORB.nextObj())　　//全ＯＲＢ上での繰り返し
{
if(PAC_ORB("TYPE")!="O")continue;　　//ＯＢＪｓの選択、ＲＥＬｓのスキップ
TsOBJ NIM_BEN(NIM,"BEN");
NIM_BEN.setObj();　　//新規ＢＥＮの作成
NIM_BEN("TITLE")=PAC_ORB("NAME");　　//属性のコピー
NIM_BEN("ALIAS")=PAC_ORB("CODE");
NIM_BEN("NOTES",1)="%PX8_CODE"+PAC_ORB("CODE");//属性のコンカテネーション
NIM_BEN("NOTES",2)="%PX8_NAME"+PAC_ORB("NAME");//ＮＯＴＥＳの第二行
NIM_BEN("NOTES")+=PAC_ORB("DOC");　　//集合体またはブロックへの添付
NIM_BEN("TITLE")+="_suffix";　　//値への連結
NIM_BEN("PROP")=Str(PAC_ORB("PROPERTY"),5,6);　　//サブストリング５、長さ６の移動
}
```

注意：ＴｓＯＢＪ＝ＴｓＯＢＪ；は、ＴｓＯＢＪオブジェクトではなく属性値をコピーすることを意味する。

【００４４】

```
TsOBJ nim_obj(nim);　　//コンストラクタクーロンＴｓＯＢＪｎｉｍをコピーする
nim_obj.copy(nim);　　//ＴｓＯＢＪｎｉｍをＴｓＯＢＪｎｉｍ＿ｏｂｊにコピーする
nim_obj=nim;　　//ｎｉｍの属性値をｎｉｍ＿ｏｂｊにコピーする
```

参照された"Include"ファイル

```
#include"tsarray.h"
#include"tsstr.h"
#include"tsrec.h"
```

Ｉ／Ｏマニピュレータ

```
#difine setleft setiosflags(ios::left)
#difine resetleft resetiosflags(ios::left)
```

マクロのステップバイステップ実行

```
extern int Tracing;
#define TRACE(arg) if (Tracing) cout << setw(5) <<_LINE_<<""<< arg<<NL;
#define COPL(c)"TsOBJ(" << c->classNdx << ","<< c->objNdx << "," << c->propNdx << "," <<c-> lineNbr << ")"
```

全体のサイズ－ＴｓＡｒｒａｙインクリメント増加

```
const int MaxClass = 128;　　//クラスの種類（オブジェクトのインスタンスなし）
const int MaxProps = 64;　　//オブジェクトあたりの属性
const int MaxKeys = 64;　　//キーブロックの要領
```

メタモデル内で定義された属性の種類

```
enum TsPropType{  Value=1,　　//シングルストリングの値
                  Array=2,　　//多重独立ストリングの値
                  Block=3};　　//連続テキストの多重ライン
extern const char* TsPropTypeStr[];　　//"enum"をストリングに変換するための配置
inline const char*cvPropType(TsPropType t)
```

27　　28

```
{
  return TsPropTypeStr[t >= Value && t <= Block ? t:0];
}
```

参照

```
class TsDynamo;
class TsPropValue;
class TsObjectInst;
class TsObjKeyArray;
class TsObjInstArray;
class TsModelObjArray;
class TsPropDefnArray;
class TsPropDefn;
class TsObjClassDefn;
class TsObjClassArray;
class TsLinkInst;
class TsLinkKeyArray;
class TsLinkInstArray;
class TsModelLnkArray;
```

ダイナモーインタフェースクラス

ＴｓＤｙｎａｍｏクラス

```
{
  friend class TsOBJ;
  friend class TsLNK;
  class TsObjClassArray*    oca;      //メタモデルのオブジェクトクラスの
表
  class TsModelObjArray*    moa;      //モデルのオブジェクト表
  class TsLnkClassArray*    lca;      //メタモデルのリンククラス表
  class TsModelLnkArray*    mla;      //モデルのリンク表
  TsDynamo(const char* metaModelName, const char* modelName);
  TsDynamo(const TsDynamo& d, const char* modelName);      //同一のメタ
モデル、新規モデル
  ～TsDynamo();      //デストラクタ
  void display();      //ダイナモ検索、表示値
  void dump();      //オブジェクト、ｉ９、属性、ｉ９、フォーマット値（Ｏ
ＢＳ）へのモデルイメージコピー
  void writeNDF(char* ndfPath);      //ダイナモ検索、表示値
  int readMetaModel(const char* fileName);      //オブジェクトの読み出し
／属性の定義
  void define(const char* obj, const char* prop, char* key,
          TsPropType prop_Type, int prop_Size);
 void freeze();
```

ＴｓＯＢＪクラスにおいては、ＴｓＯＢＪオブジェクトは、ダイナモオブジェクト内のアドレシングオブジェクトのためのポインタである。ＴｓＯＢＪ機能により、Ｎｄｘ値のいくつかまたは全てを使用してポインタテーブル＊のインデックスを行うことにより、ダイナモオブジェクトにアクセスすることができる。

【００４５】

```
classe TsOBJ  //ある値を機構D内にアドレスするためのインデックス番号
{
  TsDynamo*    dyn;  //データをダイナモ内にセットするためのポインタ
  int    classNdx;
  int    objNdx;
```

29

```
int    propNdx;
       lineNbr;
int    iterNdx;  //反復子用キー全体のインデックス
int    error;      //不良アドレス、ジャンプ
```

数値をともなうＴｓＯＢＪコンストラクタ

```
TsOBJ(TsDynamo& aDyn,
       int aClassNdx=1,
       int aObjNdx=1,
       int aPropNdx=1,
       int aLineNbr=1);
```

クラス名および属性をともなうＴｓＯＢＪコンストラクタ

```
TsOBJ(TsDynamo          & aDyn,
       const char*     aClass,
       int             aObjNdx=1,
       const char*     aProp=1,
       int             aLine=1);
```

ＴｓＯＢＪコピーのコンストラクタ

ＴｓＯＢＪ（ＴｓＯＢＪ＆　ｃ）；

ＴｓＯＢＪデストラクタ

～ＴｓＯＢＪ（）｛｝

ＴｓＯＢＪを介してダイナモオブジェクトをアドレスするための機能

```
TsStr&          getV();
TsPropValue*    getPropValue();
TsObjectInst*   getObject();
TsObjKeyArray*  getPrimeKeyArray();
TsObjKeyArray*  getSecndKeyArray();
TsObjInstArray* getObjInstArray();
TsModelObjArray* getModelObjArray();
TsPropDefnArray* getPropDefnArray();
TsPropDefn*      getPropDefn();
TsObjClassDefn*  getObjClassDefn();
TsObjClassArray* getObjClassArray();
char*           getClassName();
char*           getPropName();
```

モデル内のオブジェクトおよび属性を設定し読み出すための機能

```
int setObj(const char* aClass, int aObjNdx=1);
int setProp(const char* aProp);
int setValue(const char* aProp, int aLineNbr,const char* aValue);
int setValue(int aLineNbr, const char* aValue);
int setValue(const char* aValue){return setValue(1,aValue);}
int setValue(TsStrList* aValue);
TsStr getValue(const char* aObj, int aObjNdx, const char* aProp,int
        aLineNbr=1);
TsStr getValue(const char* aProp, int aLineNbr=1);
TsStrList getList(const char* aProp);
TsPropType getPropType();
int findObj(const char* aObj, const
char* aProp, const char* aValue);
```

ブリッジのシンタックスを改善するための作業機能

```
TsOBJ& operator()(const char* aObj,
```

31　　32

```
int aObjNdx, const char* aProp, in
t
        aPropNbr);
TsOBJ& operator()(const char* aObj, int aObjNdx, const char* aProp);
TsOBJ& operator()(const char* aObj, char* aProp);
TsOBJ& operator()(const char* aProp, int aLineNbr);
TsOBJ& operator()(const char* aProp);
TsOBJ& operator()();      //op()では不良は許されない
TsOBJ& copy(TsOBJ& b);      //ＴｓＯＢＪをコピーする
TsOBJ& operator=(TsOBJ& in);      //属性値をコピーする
TsOBJ& operator=(const char* in);      //ストリング値をコピーする
TsOBJ& operator+=(TsOBJ& in);      //属性値を添付する
TsOBJ& operator+=(char* in);      //ストリング値を添付する
int operator!(){return error;}  //最新の参照または割り当てが不良である
かどうか通報する
operator TsStr&(){return getV();}  //全てのＳｔｒ　ｏｐｓのアクティブ
ｏｐの変換
```

クラス、オブジェクト、属性、ラインを巡回するための反復機能。

【００４６】反復を停止するため、最新のオカレンスを　　＊【００４７】
超過した時０に復帰。　　＊20

```
集合体にジャンプを組み込む"null"ポインター
int firstClass();
int firstObject();
int firstProperty();
int firstLine();
int nextClass();
int nextObject();
int nextProperty();
int nextLine();
};
基数性
enum Cardn {C0_1, C1_1, C0_M, C1_M};
extern char* Cardn_str[];      //Ｃａｒｄｎをテキストに変換するための
ストリング集合体
    キー属性の値をソートするためのクラスインデックスおよびオブジェクト
インデックス
struct Clob
{
  int    classNdx;
  int    objNdx;
};
```

機構Ｄにおいては、「メタモデル」クラスは、操作すべ　　※タモデルを表す。
きモデル内のデータをインタープリットするための一定　　【００４８】
構造としての中立情報モデルＮＩＭまたは関連ツールメ※

```
属性の定義の集合体のクラスの定義
    "enum"をテキストに変換するためのストリング集合体
const char*TsPropTypeStr[]={"??","Value","Array","Block"};
classe TsPropDefn  //ＰｒｏｐＤｅｆｎＡｒｒａｙに含まれる
{
friend class TsPropDefnArray;
```

```
   ３３
  friend class TsObjectInst;
  friend class TsOBJ;
  TsStr  propName;
  TsPropType    type;
  int        length;
 TsPropDefn(const char* aPropName, TsPropType aType, int aLength);
  ～TsPropDefn() {}
};
    属性定義の集合体
TsArray(TsPropDefn*, TsPropDefnPtArray);      //ゲージを特定化する
  ＴｓＰｒｏｐＤｅｆｎＡｒｒａｙクラス
{
friend class TsObjClassDefn;
  friend class TsObjClassArray;
  friend class TsObjectInst;
  friend class TsDynamo;
  friend class TsOBJ;
  TsPropDefnPtArray      Prop;
  TsPropDefnArray() : Prop(MaxProps) {}
  ～TsPropDefnArray()      //表によって指名される各オブジェクトを削除す
る
  void display(TsOBJ* at);
  TsPropDefn* define(const char* aName, TsPropType aType, int aLength);
  TsPropDefn* insert(const char* aName, TsPropType aType, int aLength);
  int getIndex(const char* aName);
  TsPropDefn* find(const char* aName);
  };
  属性名をもとめ、表のインデックス（または０）を返す
  int TsPropDefnArray::getIndex(const char* name);
  属性名をもとめ、ＰｒｏｐＤｅｆｎ（またはＮＵＬＬ）アドレスを返す
  TsPropDefn* TsPropDefnArray::find(const char* name);
  属性名でソートされたＴｓＰｒｏｐＤｅｆｎＡｒｒａｙにＴｓＰｒｏｐＤｅｆ
ｎを挿入する
TsPropDefn*TsPropDefnArray::insert(constchar*aName,TsPropType aType,
      int aLength);
  新規ＴｓＰｒｏｐＤｅｆｎを定義する、属性名が重複しているかどうか確認す
る
TsPropDefn*TsPropDefnArray::difine(constchar*Name,TsPropTypeType,
           int Length);
  オブジェクトクラスの定義、メタモデルのオブジェクトの情報
  class TsObjClassDefn
{
  friend class TsObjClassArray;
  friend class TsObjInstArray;
  friend class TsObjectInst;
  friend class TsDynamo;
  friend class TsOBJ;
  TsStr    className;      //オブジェクトクラスの名前
  TsStr    parentName;      //類似性を有するオブジェクト名
  TsObjClassDefn*  parent;      //継承属性
```

35

```
  TsPropDefnArray* pda;      //属性名の二分法検索
  TsStr    primeKeyName;
  int      primeKeyPrNdx;
  TsStr    secndKeyName;
  int      secndKeyPrNdx;
  TsObjClassDefn(const char* Name, const char* ParentName=0,
       const char* Prime=0, const char* Second=0);
  ~TsObjClassDefn()
  void TsObjClassDefn::display(TsOBJ* at);
  メタモデルがフリーズした時、キー属性インデックスを翻訳する
};
  オブジェクトクラスの表の定義
     オブジェクトクラスの表、メタモデルの頂点
TsArray(TsObjClassDefn*, TsObjClassDefnPtArray);//ゲージを特定化する
  ＴｓＯｂｊＣｌａｓｓＡｒｒａｙクラス
{
  friend class TsOBJ;
  TsObjClassDefnPtArray    Class;
  TsStr    metaModelName;
  int    usageCount;      //共通メタモデルを保護する
  int    isFrozen;      //０＝ｆｉｎｄにより新規ｏｂｊ／ｐｒｏｐ　ｄｅｆ
ｎｓが挿入される
  TsObjClassArray(const char* name):Class(MaxClass);
  ~TsObjClassArray();
  void release() {if(--usageCount==0) delete this;}
  void display(TsOBJ* at);
  void writeNDF(TsOBJ* at, ostream& ndf);
  void resolveKeys();
  TsObjClassDefn* define(const char* name);
  TsObjClassDefn* difineKey(const char* obj, char* key, const char* prop
);
  TsObjClassDefn* insert(const char* name);
  TsObjClassDefn* find(const char* name);
  int getIndex(const char* name);
};
  類似性を有するクラス名およびキー属性のインデックスを翻訳する
  オブジェクトクラスの名前用の表のインデックスを検索する
int TsObjClassArray::getIndex(const char* name)
  オブジェクトクラスの名前により、オブジェクトクラスの定義のアドレスを検
索する
TsObjClassDefn* TsObjClassArray::find(const Char* name)
  クラス名前でソートされたオブジェクトクラスの表に、オブジェクトクラスの
定義を挿入する
TsObjClassDefn* TsObjClassArray::insert(const char* name)
  オブジェクトクラスの表の中で、オブジェクトクラスの新規定義を定義し、重
複しているかどうか確認する
TsObjClassDefn* TsObjClassArray::define(const char* name)
  第一または第二キー属性名を定義する
TsObjClassDefn* TsObjClassArray::defineKey(const char* obj,
                                char* key, const char* prop)
```

機構Ｄにおいては、モデルのデータのクラスは、対応す　＊よって定義される構造内のユーザのモデルを表す。
る中立情報モデルＮＩＭまたは関連ツールメタモデルに＊　【００４９】

属性値のクラスの定義

```
classe TsPropValue            //単一属性の値のストリング
{
  friend class TsObjectInst;
  friend class TsOBJ;
  TsPropValue* next;          //多価属性が連鎖される
  int        lineNbr;          //挿入用のソートファイル
  TsStr  v;                    //ストリングの割り当ては再寸法決定をサポ
ートする
  TsPropValue(int line_Nbr, const char* val)
  ～TsPropValue(){}
  void display(TsOBJ* at);
  void dump(TsOBJ* at);
  void writeNDF(TsOBJ* at,ostream& ndf);
  void set(int line_Nbr, const char* val);
  TsStr get(int line_Nbr);
  operator const char*()const{return (const char*)v;}
};
  void TsPropValue::display(TsOBJ* at)
  void TsPropValue::dump(TsOBJ* at)
  void TsPropValue::writeNDF(TsOBJ* at, ostream& ndf)
      ライン番号、および属性値用の値を設定する
void TsPropValue::set(int line_Nbr, const char* val)
      多重ライン属性値を設定する
void TsPropValue::set(const TsStrList* val)
      ライン番号で属性値を読み出す
TsStr TsPropValue::get(int line_Nbr)
      多重ライン属性値を読み出す
TsStrList TsPropValue::get()
```

オブジェクトのインスタンスのクラスの定義

　　オブジェクトのインスタンスは属性のインスタンスの表を含み、属性値を指定する。

【００５０】

```
TsArray(TsPropValue*,  TsPropValuePtArray);//spcialise騙@le gabarit
      ゲージを特定化する
class TsObjectInst  //ＰｒｏｐＤｅｆｎＡｒｒａｙと平行なインデックス
{
  friend class TsObjInstArray;
  friend class TsOBJ;
  TsPropValuePtArray   Prop;
  TsObjectInst() : Prop(MaxProps){}
  ～TsObjectInst()
  void display(TsOBJ* at);
  void dump(TsOBJ* at);
  void writeNDF(TsOBJ* at,ostream& ndf);
  TsPropValue* setValue(TsOBJ* at, const char* val);
};
```

オブジェクトのキー集合体のクラスの定義

３９

キー属性を使用するオブジェクトキーおよびオブジェクトインデックスの集合体

```
TsArray(int, TsObjNdxArray);  //ゲージを特定化する
class TsObjKeyArray          //キー属性の値でソートされたＯｂｊＮｄｘｓ
{
  friend class TsObjInstArray;
  friend class TsOBJ;
  TsObjNdxArray      objNdx;    //命令内に送信または二分法検索
          dichotomique
  TsObjKeyArray():objNdx(MaxKeys){}
  ～TsObjKeyArray(){}
  void display(TsOBJ* at);
  void insert(TsOBJ* at);
  int find(char* val);
};
```

オブジェクトのインスタンスの表のクラスの定義

```
TsArray(TsObjectInst*,TsObjectPtArray);  //ゲージを特定化する
class TsObjInstArray          //ＯｂｊＮｄｘによりインデックスされたオブジェクトのインスタンス
{
  friend class TsModelObjArray;
  friend class TsOBJ;
  TsObjKeyArray*  primeKeyArray;
  TsObjKeyArray*  secndKeyArray;
  TsObjectPtArray  Obj;
  TsObjInstArray():Obj(MaxKeys)
  ～ＴｓＯｂｊＩｎｓｔＡｒｒａｙ()
```

オブジェクトインデックスを位置決めし、必要であればオブジェクトインデックスを作成する

```
  ＴｓＯｂｊｅｃｔＩｎｓｔ*  ｓｅｔＯｂｊｅｃｔ(ｉｎｔ  ｏｂｊＮｄｘ)
  void display(TsOBJ* at);
  void dump(TsOBJ* at);
  void writeNDF(TsOBJ* at, ostream& ndf);
  void updateKey(TsOBJ* at);
};
```

キー属性値に従う挿入によるオブジェクトのキーの集合体の更新

```
void TsObjInstArray::updateKey(TsOBJ* at)
```

所与の値に現在の属性を設定する。

【００５１】

```
TsPropValue* TsObjectInst::setValue(TsOBJ* at, const char* val)
```

モデルのオブジェクトの表のクラスの定義

モデルのオブジェクトの集合体、モデルの頂点、オブジェクトのインスタンスの名前

```
TsArray(TsObjInstArray*, TsObjInstPtArray);  //ゲージを特定化する
class TsModelObjArray                        //ｃｌｎｄｘによってインデックスされるオブジェクトのクラス
{
  friend class TsOBJ;
  TsStr      modelName;
  TsObjInstPtArray Class;
```

```
public:
  TsModelObjArray(const char* Name):Class(MaxClass)
  ~TsModelObjArray()
  void display(TsOBJ* at);
  void dump(TsOBJ* at);
  void writeNDF(TsOBJ* at, ostream& ndf);
  Positionner le tableau courant d'instance d'objets avec L'index de c
lasse TsObjInstArray* setClass(int classNdx)
};
```

クラスのインデックスによりオブジェクトのインスタンスの現在の表を位置決めする

機構D内のリンクのクラスの定義は、オブジェクトのクラスの定義のサブクラスである。モデルのリンクはリンクのインスタンスである。リンクのクラスの定義は、リンクが関連付けることができるオブジェクトのクラスを記述する。リンクのインスタンスは、関連付けられたモデルのオブジェクトのインスタンスを示す。

【００５２】リンクのオブジェクトは通常オブジェクトを記述する。なぜなら、リンクはオブジェクトの特徴を有する（名前をもち、属性をもつことができる）一方、「ｔｏ」および「ｆｒｏｍ」のベクトルおよび基数性も有するからである。パフォーマンスを最適化する目的から、リンクに付加されるデータ要素は、オブジェクトの属性として使われるよりもむしろＣ＋＋言語で符号化される。ソートされたインデックスの集合体によっても検索を最適化することができる。

【００５３】機構Dのコンストラクタは、ツールのメタモデルまたは、中立情報モデルＮＩＭのメタモデルファイルをもとにして、別のオブジェクト定義とのリンクの定義を読む。

【００５４】リンクは、リンクによって接続されるモデルオブジェクトを見つけ出すことができるようにするため、前記オブジェクトからのおよび前記オブジェクトへのオブジェクトインデックスｃｌＮｄｘおよびｏｂｊＮｄｘを含む。リンクの定義は、定義によってまとめることができるオブジェクトを指定するが、実クラスは指定されたオブジェクトクラスから派生することができる。従って、リンクはインデックスｃｌＮｄｘおよびｏｂｊ＊Ｎｄｘを含む。リンクはオブジェクトと同様、属性をもつことができる。

【００５５】二つのオブジェクト間にリンクをつくるということが、インデックス表にリンクを入力することも意味する。全オブジェクト、またはある種のリンクがすでにまとめている特定のオブジェクトを見つけ出すためには、リンクのキーの集合体がソートされる。全リンク、またはあるオブジェクトからのまたはあるオブジェクトへの特定のリンクを見つけ出すためには、オブジェクトリンクの集合体がソートされる。リンクのクラスの定義は、数値がオブジェクトのリンクを通しての反復順序を決定する属性のインデックスｐｒｏｐＮｄｘを含むことができる。これにより、特定の順序で、エンタイティーの属性および属性のサブフィールドを調べることができる。リンクのインデックスの集合体のデフォルトシーケンシングは、ターゲットオブジェクトのキーを使用するので、二分法検索およびリストの並べ換えが簡単になる。

【００５６】あるメタモデルのモデルと別のメタモデルのモデルとの対応関係をとることができるブリッジ構造は、実際には、モデルの各一次オブジェクトを調べるために反復子を使用し、次に、組み合わされたオブジェクトを見つけ出すためリンクを走るかストレンジャキーを使用するループの階層を利用する。すでにインプリメントされているかコピーされているモデルＮＩＭを変更する他の処理は、完全性および他のサービスの作業の確認のために、同じ種類の一般的処理を使用する。

【００５７】

```
#include "tsdynamo.h"
//"Cardn enum"の値とテキストとの対応関係をとるためのストリングの集合体
char* Cardn_str[]={"0:1","1:1","0:m","1:m"};
  リンクのクラスの定義
classe TsLinkClassDefn:TsObjClassDefn
{
  friend class TsLNK;
  friend class TsLinkClassArray;
  TsStr  fromVerb;
 TsStr  toVerb;
  int    fromCINdx;
  int    toCINdx;
```

```
  43
  Cardn  fromCardn;
  Cardn  toCardn;
  int    toKeyPropNdx;      //オブジェクトのリンクの集合体の順序化
  TsStr  toSetMbr;
  TsStr  fromSetMbr;
  TsLinkClassDefn( const char* aClassName,
                   const char* aFromVerb,
                   const char* aToVerb,
                   const char* aFromClName,
                   const char* aToClName,
                   const char* aFromCardn,
                   const char* aToCardn,
                   const char* ToKeyPropName,
                   const char* aToSetMbr,
                   const char* aFromSetMbr);
  void display(TsLNK* at);
};
void TsLinkClassDefn::display(TsLNK* at);
```

モデルのレベルでのリンク

ＴｓＬｉｎｋＣｌａｓｓＤｅｆｎクラス：

```
classe TsLinkInst: TsObjectInst
{
  int      fromCINdx;
  int      fromObjNdx;
  int      toCINdx;
  int      toObjNdx;
 TsLinkInst(TsOBJ& aFrom,TsOBJ& aTo);
  ～TsLinkInst(){}
  void desplay(TsLNK* at);
  void putNDF(ostream& ndf,TsLNK* at);
  TsPropValue* setProp(TsLNK* at, const char* val);
};
void TsLinkInst::display(TsLNK* at);
```

リンクのキー集合体、キー属性を使用するソートされたリンクにインデックス

```
TsArray(TsClob, TsClobArray); //ゲージを特定化する
classe TsLinkKeyArray                    //ターゲットオブジェクトのキー
```

属性の値によってソートされたＩｎｋＮｄｘｓ

```
{
  friend class TsLinkInstArray;
  friend class TsLNK;
  TsClobArray              linkNdx; //命令内に送信または二分法検索
  TsLinkKeyArray():linkNdx(MaxKeys){}
```

ＴｓＬｉｎｋＩｎｓｔクラス：

```
  ～TsLinkKeyArray()
  voiddisplay(TsLNK* at);
  void insert(TsLNK* at);
  int find(char* val);
};
```

リンクのキーの集合体の更新、キー属性値に従う挿入

```
void TsLinkInstArray::updateToKey(TsOBJ* at);
```

リンクのキーの集合体をもとにした更新、キー属性値に従う挿入
void TsLinkInstArray::updateFromKey(TsOBJ* at);
ＴｓＬＮＫクラス、インタフェースクラス
ＴｓＬＮＫクラスはＴｓＯＢＪクラスから派生する。Ｔ　＊ナビゲーションである。
ｓＬＮＫクラスの付加的な可能性とは、メタモデル内の　【００５８】
リンクを介するモデル内のオブジェクト間の検索または＊
現在のリンクを選択する
int TsLNK::setLink(const char* aClass, int aLnkNdx);
//所与のインデックスに固有なリンクを設定する
//必要であれば、リンクのツリーを増やす
int TsLNK::setLink(TsOBJ& from, TsOBJ& to);
現在のリンクの初期状態へのリセット
int TsLNK::resetLink(TsOBJ& from, TsOBJ& to);
固有リンクのターゲットを読み出す
TsClop TsLNK::getTarget();
固有リンクの出所（ソース）を読み出す
TsClop TsLNK::getSource();
必要とされるリンク種類についてのリンクのインデックスおよびキー値を求める
int TsLNK::findLink(const char* aLnk, const char* aProp, const char* aVal);
現在のオブジェクトのインデックスを、そのベクトルインデックス集合体に挿入する
void TsVtr Key Array::insert(TsOBJ*at)

次に、種々のツールエキスポートファイルフォーマットのレコードの符号化および解読に関するクラスに定義を示す。

【００５９】可変長フィールド用のレコード記述子
ＣおよびＣ＋＋言語ではレコードを完全にはサポートすることができない。これら言語は、文字集合および下部構造など、データ要素をともなう文字集合および構造を提供する。ある構造が宣言された後は、プログラマはプロシジャコードの内容にアクセスするために、このコード内において再度、長さについての情報を符号化しなければならない。デリミッタを使用する可変長フィールドは通常、レコードの構文分析を行うコード内のレコードの構文の符号化により処理される。今回の場合提案され定義される有利な解決方法は、フィールド記述子のコレクションとしてレコードの宣言を使用することである。まず、プログラマが固定および可変サイズのフィールドを定義し、レコードの種類に一定の値のコードおよびデリミッタ値を割り当てる。次にプログラマはプロシジャコード内で、レコードとその記述子との対応関係をつくるための機能を呼び出し、次にレコード内のフィールドにアクセスする。情報を出力するためにプログラマは、※

```
struct TsFieldDescr
{
 TsFieldType      type;
 char*  name;
 int      length;
```

※レコードをデリートしそのフィールドに入力し次に出力のため完全なレコードを供給することができる機能を呼び出す。レコードの記述には、実行時の情報の名前、種類およびサイズが含まれる。記述情報およびデータはデバッギングのために表示することができる。

【００６０】ＴｓＲｅｃクラスを使用する場合、ｔｓｒｅｃ．ｈファイルは、レコードのフィールドからのデータを供給し、レコードのフィールド内にデータを設定し、出力ができるよう完全なレコードを供給することにより、レコード記述子作成のためのマクロ、およびレジスタとレコード記述との対応関係をとるための機能のプロトタイプを定義する。

【００６１】このようにＴｓＲｅｃクラスは、機構Ｄの多数のファイルおよび、ブリッジの構成要素によって読み書きされる多数のツールエキスポートファイルを処理することができる。

【００６２】フィールドの種類を定義する
enum TsFieldType{REC,BFR,LIT,FLD,VAR,END};
レコードの記述は構造の表である。

【００６３】

47 48

```
};
  プログラマがレコードの記述を定義するのを支援するためのマクロ
#define MaxRec(len) const int RecBufSize=len;
#define XRec(name) extern char RECBUF##name[];¥
                extern TsFieldDescr name[];
#define Rec(name)  char RECBUF##name[RecBufSize+1];¥
                TsFieldDescr name[]={{REC, #name, 0},¥
                                {BFR, RECBUF##name, RecBufSize},
#define Lit(string)  {LIT, string}
#define  Fld(name,  len)  {FLD,  #name,  len}
#define  Var(name)  {VAR,  #name}
#define  End      {END}}
  フィールド値にアクセスするためのマクロ
#define  GetFld(rec,fld)  getfld(rec,  #fld)
#define  SetFld(rec,fld,val)  setfld(rec,  #fld,  val)
  機能のプロトタイプ
  整数を、基底、サイズ、およびフィルのオプション付きＡＳＣＩＩに変換する
char*itoa(int value, int base=10, int size=0, char fill='');
  ＡＳＣＩＩを整数に逆変換
int atoi(const char* p);
  動的レコード機能
  レコードとのレジスタのマッチング
int match(char* buffer, TsFieldDescr* f);
  レコードのフィールドの読み出し
TsStr getfld(const TsFieldDescr* rec, const char* name);
  ｃｈａｒ*値をともなうフィールドの読み出し
ｖｏｉｄ　ｓｅｔｆｌｄ（ＴｓＦｉｅｌｄＤｅｓｃｒ*　ｒｅｃ，　ｃｏｎｓｔ
  ｃｈａｒ*　ｎａｍｅ，　ｃｏｎｓｔ　ｃｈａｒ*　ｖａｌｕｅ）；
  ＴｓＳｔｒクラス（後で定義する）の値をともなうフィールドの読み出し
ｉｎｌｉｎｅ　ｖｏｉｄ　ｓｅｔｆｌｄ（ＴｓＦｉｅｌｄＤｅｓｃｒ*　ｒｅ
ｃ，　ｃｏｎｓｔ　ｃｈａｒ*　ｎａｍｅ，　ｃｏｎｓｔ　ＴｓＳｔｒ&　ｖａ
ｌ
ｕｅ）；
  ｃｈａｒ*に変換された整数値をともなうフィールドの読み出し
inline void setfld(TsFieldDescr* rec, const char* name, const int value)
{
  setfld(rec, name, itoa(value));
}
  レコードを消去する
void clear(TsFieldDescr* rec);
  レコードを表示する
void display(const TsFieldDescr* rec);
XRec(IO_TAF);
  "enum TsFieldType"タイプをｃｈａｒ*シンボルに変換する
char* aTsField Type(const TsField Typet;)
  レコード名のポインタの読み出し
inline char* GetRec(const TsFieldDescr* rec){return(rec+1)->name;}
  外部になるようＩＯ_ＴＡＦを宣言する
XRec(IO_TAF);
```

また、文字ストリングの取り扱いに関するクラスの定義＊50＊も示す。

４９

【００６４】文字ストリングクラスＴｓＳｔｒの定義ヘッダーファイルｔｓｓｔｒ．ｈは、きわめて有利な文字ストリングのデータ種を提供するＴｓＳｔｒクラスの定義を含む。なぜならこの種は、Ｃ言語に内蔵されている多数のデータ種のような挙動をもつからである。同じく重要なことは、通常、メモリブロックの損失に関わる情報の損失が一切禁止されることである。　　*

５０

*【００６５】ＴｓＳｔｒクラスの目的は、Ｃ型言語のｃｈａｒ*文字ストリングとの互換性を保ちつつ、割り当て、コンカテネーション、サブストリング機能との比較およびメモリの動的再割り当て演算子をサポートできるＣ＋＋文字ストリングクラスを定義することである。【００６６】

```
  ライン文字の新規マクロ
#define NL "¥n"
#define NL2 "¥n¥n"
      名前付き変数の表示
#define SEE(x) " "#x": "<<x
//                              Usage:cout<<SEE(abc)<<
          SEE(xyz);
      文字ストリングのクラス
  ＴｓＳｔｒクラス
{
  char*  pt;//動的に割り当てられた文字の集合体
  size_t  ln//現在のデータ長
  size_t  bs;      //ブロック容量（ストップビットは数えず）
  デフォルトブロックサイズをもつ空ＴｓＳｔｒを設定する
void Defaults(){bs=16; ln=0;pt=new char[bs+1]; pt[0]='¥0';}
public:
  ｃｈａｒ*文字ストリングによって初期化されるＴｓＳｔｒクラスを作成する
TsStr(const char* cp);
  ＴｓＳｔｒによって初期化されるＴｓＳｔｒクラスを作成する
TsStr(const TsStr& b);
  デフォルトＴｓＳｔｒクラスを作成する
TsStr(){Defaults();}
  あるデータブロックサイズをもつ空のＴｓＳｔｒクラスを作成する
TsStr(int len);
  ｃｈａｒ*文字ストリングおよび長さはわかっているので、ＴｓＳｔｒクラス
を作成する
TsStr(const char * cp, const int len);
  デストラクタ：文字全てをデリートする
~TsStr(){delete[]pt;}
  ＴｓＳｔｒ＝ａ　ＴｓＳｔｒクラスを割り当てる
  TsStr& operator=(const TsStr &b);
  ＴｓＳｔｒ＝ａ　ｃｈａｒ*　ｓｔｒｉｎｇクラスを割り当てる
  TsStr& oprator=(const char* bpt);
騙@ＴｓＳｔｒクラス＝一文字を割り当てる
  TsStr& oprator=(char b);
騙@ＴｓＳｔｒクラス＋ＴｓＳｔｒクラスを連結する
  TsStr& oprator+(const TsStr &b)const;
騙@ＴｓＳｔｒクラス＋ｃｈａｒ*文字ストリングを連結する
  TsStr& oprator+(const char* bpt)const;
騙@ＴｓＳｔｒクラスをＴｓＳｔｒクラスに添付する
  TsStr& oprator+=(const TsStr &b);
騙@ｃｈａｒ*文字ストリングをＴｓＳｔｒクラスに添付する
  TsStr& operator+=(const char* bpt);
```

```
  一文字をＴｓＳｔｒクラスに添付する
  TsStr& operator+=(char b);
  ＴｓＳｔｒクラスをｃｈａｒ*文字ストリングに変換する
  operetor char* (){return pt;}          //変換演算子
  ＴｓＳｔｒクラスの一定リファレンスを一定ｃｈａｒ*文字ストリングに変換
する
operator const char*()const{return pt;}  //変換演算子
  ＴｓＳｔｒクラスをＴｓＳｔｒクラスと比較する
int  operator==(const  TsStr  &b)const{return  strcmp(pt, b.pt)==0;}
int  operator  !=(const TsStr  &b)const{return  strcmp(pt, b.pt)!=0;}
int operator >(const TsStr &b)const{return strcmp(pt, b.pt)> 0;}
int  operator>=(const  TsStr  &b)const{return  strcmp(pt, b.pt)>=0;}
int operator< (const TsStr &b)const{return strcmp(pt, b.pt)< 0;}
int  operator<=(const  TsStr  &b)const{return  strcmp(pt, b.pt)<=0;}
  ＴｓＳｔｒクラスをｃｈａｒ*文字ストリングと比較する
int operator==(const char*b)const{return strcmp(pt, b)==0;}
int operator !=(const char*b)const{return strcmp(pt, b) !=0;}
int operator> (const char*b)const{return strcmp(pt, b)> 0;}
int operator>=(const char*b)const{return strcmp(pt, b)>=0;}
int operator< (const char*b)const{return strcmp(pt, b)< 0;}
int operator<=(const char*b)const{return strcmp(pt, b)<=0;}
  ｃｈａｒ*文字ストリングをＴｓＳｔｒクラスと比較する
friend int operator==(const char*a, const TsStr &b){return strcmp(a,b.pt
)==0;}
friend int operator!=(const char*a, const TsStr &b){return strcmp(a,b.pt
)!=0;}
friend int operator> (const char*a, const TsStr &b){return strcmp(a,b.pt
)> 0;}
friend int operator>=(const char*a, const TsStr &b){return strcmp(a,b.pt
)>=0;}
friend int operator< (const char*a, const TsStr &b){return strcmp(a,b.pt
)< 0;}
friend int operator<=(const char*a, const TsStr &b){return strcmp(a,b.pt
)<=0;}
  ＴｓＳｔｒクラスのためのストリング出力演算子
  friend ostream& operator <<(ostream& s, const TsStr& a){return s << a.
pt;}
  ＴｓＳｔｒクラスのためのストリング入力演算子ーライン毎
  friend istream& operator >>(istream& s, TsStr& a);
  ＴｓＳｔｒクラスのｃｈａｒ*文字ストリングのポインタを読み出す
  friend char* getPt(const TsStr& s){return s.pt;}
  ＴｓＳｔｒクラスのストリング長を読み出す
  friend int strlen(const TsStr& s){return s.ln;}
  あるオフセットで、ＴｓＳｔｒクラスの一文字を読み出す
  friend char ch(const TsStr& str, int offset){return str.pt[offset];}
};//  ＴｓＳｔｒクラスの定義終了
```

本発明の環境下で使用されるクラスを含むクラスおよびファイルの記述が続く。

【００６７】ＴｓＦｉｌｅ入出力クラスの定義

ヘッダーファイルｔｓｆｉｌｅ．ｈは、ＴｓＦｉｌｅ、＊50ＴｓＦｉｌｅＲｅａｄｅｒ、ＴｓＦｉｌｅＷｒｉｔｅｒ、ＴｓＤａｔａＲｅａｄｅｒ、ＴｓＤａｔａＷｒｉｔｅｒクラスの定義を含む。これらクラスにより、ファイルの入力／出力作業をカプセル化することができる。

53

【0068】ＴｓＦｉｌｅクラスにより、ファイルポインタを利用するＵＮＩＸ（もっぱらX／ＯＰＥＮ社を介してライセンス供与される登録商標）システムの呼び出しをカプセル化することができ、ファイルの開閉はコンストラクタおよびデストラクタにより処理されるので、＊

54

＊出願人の仕事は大幅に簡略化される。

【0069】このカプセル化により作成されるその他のクラスにより、一度に一ラインすなわち一度にファイル全体の読み書きを行うことができる。

【0070】

```
ＴｓＦｉｌｅクラス
{              //
FILE * const  fp:      //ファイルポインタ
TsFile(      //コンストラクタ
  const char  *path,      //ファイルアクセス名前
  const char  *mode = "w"      //開モード
  );             //
~TsFile();      //デストラクタ
int flush();      //待ち行列過負荷（ｓ）
char *gets(char *s);      //読み出し時過負荷（ｓ）
int puts(char *s);      //書き込み時過負荷（ｓ）
int putc(char c);      //書き込み時過負荷（ｃ）
};                 //ＴｓＦｉｌｅクラス終了
                   //
Ts FileReaderクラス      //ファイルのラインの読み出し
  {
  TsFile    *f;
  char      *buffer;
  TsFileReader(      //コンストラクタ
    const char  *path,
    int       buffer_size
    );
  ~TsFileReader();      //デストラクタ
  char *getnext();      //次のデータを読み出す
  };
ＴｓＦｉｌｅＷｒｉｔｅｒクラス  //ラインをファイル内に書き込む
  {
  TsFile    *f;
  enum{NEW, APPEND};
  TsFileWriter(      //コンストラクタ
    const char  *path,
    int       new_or_append
    );
  ~TsFileWriter();      //デストラクタ
  int putnext(char *buffer);      //次のデータを書き込む
  int putnewline();      //新規ラインの文字を書き込む
  };
ＴｓＤａｔａＲｅａｄｅｒクラス
  {
  Singly_linked_list(TsStr)  datalist;
  TsDataReader();      //コンストラクタ
  ~TsDataReader();      //デストラクタ
  void loadfile(const char *path);      //ファイルをリスト内にロードする
  Singly_linked_list(TsStr)&getlist();      //リスト内を読む
  void print();      //リストを印刷する
```

55　　56

```
};
ＴｓＤａｔａＷｒｉｔｅｒクラス
{
Singly_linked_list(TsStr)  datalist;
TsDataWriter();      //コンストラクタ
～TsDataWriter();      //デストラクタ
void setlist(Singly_linked_list(TsStr)& L);      //書き込みのためにリ
ストを編成する
void dumpfile(const char *path);      //ファイルからリストを出力する
void print();      //リストを印刷する
};
```

中立データファイルＮＤＦに関しては、そのフォーマットにより、モデルを連続的に保存することができる。ファイルＮＤＦは中立情報モデルＮＩＭまたはツールメタモデルの形態でモデルを表すことができる。ファイルＮＤＦにより、属性値とともに、モデルの表示ラインのクラス、オブジェクト、属性および座標を保存することができる。

【００７１】機構Ｄのオブジェクト（ダイナモオブジェクトと呼ばれる）は、モデルを保存することができる、＊ファイルＮＤＦ内への書き込み機能を有する。ファイルＮＤＦの要求により編集が可能であることは、テストデータを作成する際に有益である。ＮＤＦフォーマットは、テスト実行時の結果を評価する場合に便利である。ブリッジの開発者は、結果を保存または評価するため、ならびにブリッジの処理の際、種々の中間点においてテストを追跡し再実行するためにファイルＮＤＦを使用することができる。

【００７２】

```
テキストの構文分析のための入力ファイルのレジスタ
ＴｓＢｕｆｆｅｒクラス
{
  char*buf;      //レジスタのポインタ
  char* cp;      //現在の文字の位置
  int  bufSize;      //レジスタ容量
  int  lineNbr;      //入力ファイル内の現在のライン番号
  int  indent;      //スペースおよび／またはスペースとして数えるタブ（
1から８）
  int  rescan;      //１：読み出しスキップ、レジスタの同内容を使用する
  FILE* fd;      //コンストラクタによって開かれるファイル
  TsBuffer(const char*aFilePath, int aLen = 256);
  ～TsBuffer(){delete[]buf;}
  テキストおよび値とレジスタの現在位置との対応関係確立、ポインタの移動
int TsBuffer::match(const char* v)
注釈ライン用テスト
int TsBuffer::isComment()
インデント検索、タブ調整
int TsBuffer::scanIndent()
テキストライン、無視された注釈の読み出し、インデントのカウント、ボーダ
ー＼ｎの設定
int TsBuffer::readBuf()
「delim」によって限定された語の読み出し
char* TsBuffer::getWord(char delim)
小数値の検索、整数値への値の返し、カーソル移動
intTsBuffer::scanDec()
ＮＤＦモデルの読み出し、ダイナモオブジェクトの作成
ＴｓＲｅａｄＮＤＦクラス：ＴｓＢｕｆｆｅｒレジスタはプロテクトされる
{
  TsStr  metaModelName;
```

57　　58

```
    TsStr  modelName;
    TsStr  objName;
    int    objNdx;
    TsStr  propName;
    int    lineNbr;
    char*  value;
    TsDynamo*dyn;
    TsOBJ* at;
    int status;
    TsReadNDF(const char*aFileName);
};
```

ＮＤＦファイルの読み出しおよびダイナモオブジェクトのロード
コンストラクタは全体のアルゴリズムである。

【００７３】TsReadNDF::TsReadNDF(constchar*aFileName):TsBuffer(aFileName)
ＮＤＦファイルのシンタックス
ＮＤＦファイルのシンタックスにおいては、インデントはネスティングを示す。注釈ラインは「*」で始まる。

```
*NDF rev 1.0      //ファイルを識別するための注釈ライン
MetaModelName:metaModelName      //モデルの読み出しに必要なメタモデルの名前
Model: modelName      //ダイナモオブジェクトを作成する（モデルの名前、メタモデルの名前）
Object: Class,ObjNdx      //オブジェクトを作成する（クラス、ＯｂｊＮｄｘインデックス）
propId: value      //単一値の属性を設定する
propsym:      //全体またはブロック内で多重値を設定する
propNdx: value...      //ラインあたり一つの値
Link: InkName,InkNdx      //リンクを作成する（名前およびリンクのインデックス）
FROM: Class,objNdx      //最終点を設定する
TO: Class, objNdx      //最終点を設定する
//"metaModelName.mta"ファイルは、中立情報モデルＮＩＭ、ＰＸ８などのメタモデルを供給する。
```

【００７４】ＮＤＦファイルのメタモデル
ＮＤＦファイルの以下のフォーマットにより、メタモデル、モデル、オブジェクト、属性および値、ならびにモデルを構成するリンクを指定することができる。

【００７５】

【００７６】//従って"modelName.ndf"ファイルには構造がない。

【００７７】//例：

```
*NDF rev 1.0
MetaModelName: nim
Model: ndfsample
Object: BEN,1
TITLE: My Object X
NOTES:
1: %PX8_CODE myobjx
2: %PX8_NAME My Object X
Object: BEN,2
TITLE: another obj
NOTES:
1: %PX8_CODE nother
2: %PX8_NAME another obj
※Object: ATT,1
  DATY: DATE
  NOTES:
  1: %PX8_CODE bdate
  2: %PX8_NAME birthdate
  PIC: YY-MM-DD
  SIZE: 8
  TITLE: birthdate
  Link: BEN1.1
  FROM: BEN,2
  TO: ATT,1
```

結論として、ブリッジをつくるために、インタープリットされるプロプリエタリ言語ではなく実行言語（例えばＣ＋＋型の）を使用する、本発明によるデータモデルの取り扱い方法についての前記説明により、複数のソフトウェア工学ツールのインターオペラビリティを可能にす

る新しい技術に応用されるようになっている前記方法の有利な技術的効果を明らかにすることができる。提供する中立情報モデルにより、あらゆるツールを簡単に表すことができるだけでなく内部モデルの表現も可能であるので、ユーザが開発データを内部的に表現するためのみずからの情報モデルを保存しておくこと、あるいは、共通リファレンシャルに組み合わされる種々のソフトウェア工学ツール（ＣＡＳＥ）用として設計された例えばデータ交換フォーマット（ＣＤＩＦ）など、産業界のあらゆる標準を利用することができる。コードの再利用性が十二分に活用されるので、このようなブリッジの発売開始までの納期に関する制約を満たすことができる一方、コードライブラリは再利用可能であるので、前記ブリッジの開発時間およびコストはきわめて顕著に削減される。同様に、あるモデルからクライアントの個別のニーズへの移行の過程を進展させきめ細かな対応をはかるためのブリッジの新規構成要素の追加プロシジャが大幅に簡略化される。本方法は、単純かつ単一の実行可能命令の応用と考えることができることから、性能がきわめて著しく向上する。

【図面の簡単な説明】

【図１】種々のソフトウェア工学ツール間の本発明によるインタオペラビリティーの例を示す図である。

【図２】本発明を実施することができる種々の構成要素が示されるアーキテクチャを示す図である。

【符号の説明】

Ｂａ、Ｂｂ、Ｂｃ、Ｂｄ　ブリッジ

ＮＩＭ　中間フォーマット

Ｔａ、Ｔｂ、Ｔｃ、Ｔｄ　ソフトウェア工学ツール

【図１】

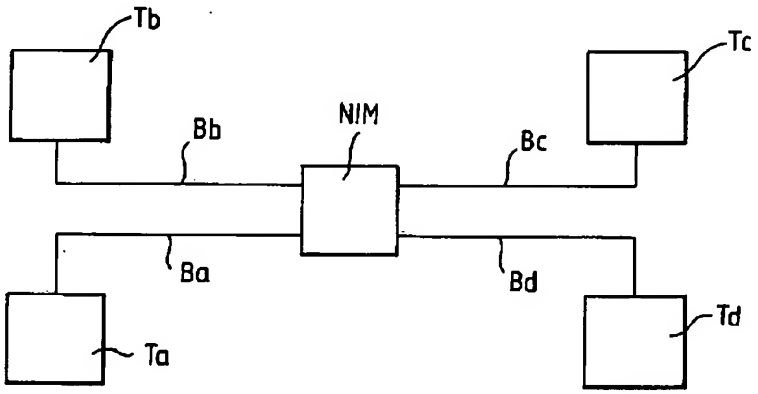

FIG.1

【図２】

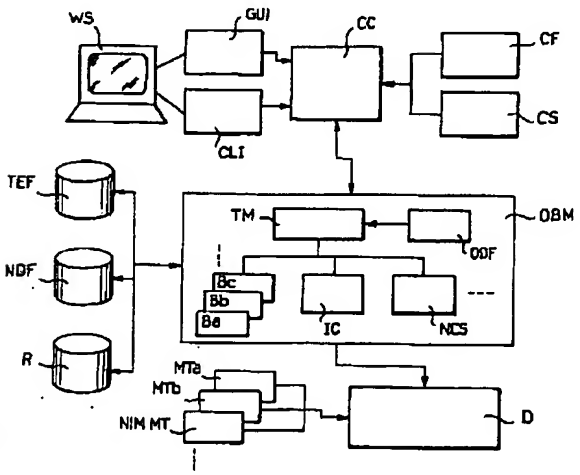

FIG.2

フロントページの続き

(72)発明者　ジエイムズ・キソ
アメリカ合衆国、マサチユーセツツ・
01720、アクトン、グレート・ロード・
38・エイ、ナンバー・301

(72)発明者　エドワード・ストラスバージヤー
アメリカ合衆国、マサチユーセツツ・
01821、ビレリカ、アンジエラ・レーン・
23